<http://web.tuat.ac.jp/~katfuji/ANAC2020/cfp/scml_cfp.pdf>

SCML:

* המשימה בגדול: לבנות סוכן אוטונומי שמבצע משא ומתן בשם מנהל מפעל, כחלק מסימולציה של ניהול שרשרת אספקה.
* המטרה: למקסם את הרווח בהינתן יכולות הייצור המוגבלות (שמתגלות בתחילת הסימולציה), ע"י משא ומתן עם סוכנים אחרים.

עוד פרטים:

* אפשר להיות בכמה משאים ומתנים בו-זמנית, כאשר היוטיליטי של המשאים ומתנים תלויים זה בזה
* מו"מ נעשים כדי לקנות מוצרים לתהליך הייצור, וכדי למכור את התוצרים.
* כל המו"מים לפי פרוטוקול alternating offers:
  + צד אחד מציע bid
  + הצד השני יכול:
    - לקבל את ההצעה
    - לדחות את ההצעה ולהציע הצעה חדשה
    - לדחות את ההצעה וללכת, ללא השגת הסכמה בין הצדדים.
  + התהליך חוזר על עצמו עד שמגיעים להסכמה/דדליין (זמן מסוים/איטרציה שאם עד אליו אין הסכמה – המו"מ נכשל).
  + כדי להגיע להסכמה, 2 הצדדים צריכים לקבל את ההצעה.
* סימולציה בודדת רצה כמות קבועה של סיבובים עם זמן מוגבל של 2 שעות.
* לא ניתן ללמוד מסימולציות קודמות. כל סימולציה היא דף חלק
* הסוכן יכול לצבור מידע על הסוכנים האחרים במהלך הסימולציה.

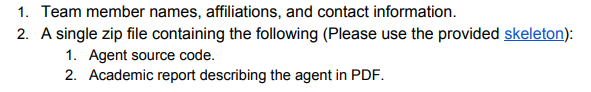
פלטפורמה:

* NegMAS – פלטפורמה למו"מ מבוססת פייתון.
* צריך להתקין ספריית פייתון scml שמכילה את סביבת SCML2020World ותרוץ על NegMAS

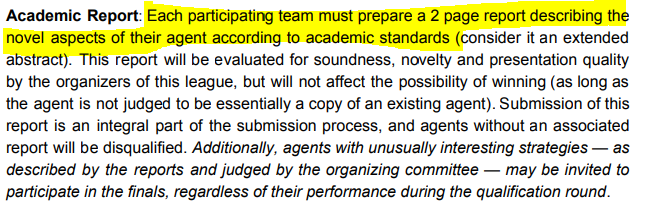
# עשיתי virtual environment והתקנתי negmes, scml.

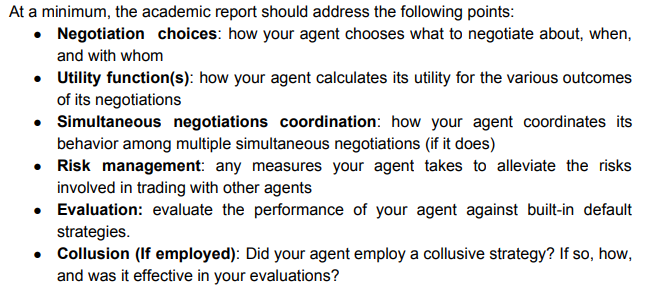
התחרות מתרחשת בלייב כבר מפברואר. אפשר להעלות גרסאות ראשוניות של סוכנים. לאתר זה יש להעלות גם את הגרסה הסופית של הסוכן.

להעלות לאתר (<https://scml.cs.brown.edu/>):



מבחינה אקדמית יש להכין:





* ייתכן שישימו את הסוכן שלנו במו"מ מול עצמו
* ה score של הסוכן זה חציון ה profit של כל האינסטנסים שלו מכל הסימולציות

<http://www.yasserm.com/scml/scml2020.pdf>

על הסימולציה:

* העולם: מפעלים שקונים ומוכרים מוצרים אחד מהשני.
* המפעלים מיוצגים ע"י סוכנים, שהם בעצם מנהלי המפעל.
* כל סוכן בוחר עם מי לבוא במו"מ במכירה ובקנייה.
* כל סימולציה מסתיימת ב profit
* הסוכן עם ה profit הממוצע (או חציון?) הכי גבוה – זוכה.

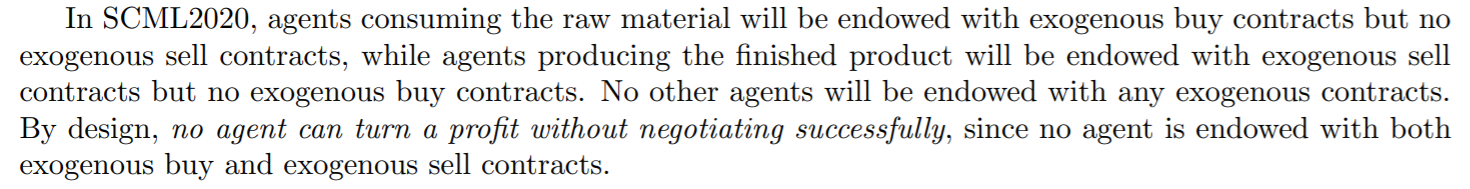
המפעלים:

* ממירים מוצרים מסוג אחד למוצרים מסוג אחר בתהליך שנקרא manufacturing, ושנעשה ב production lines
  + ייצור לוקח יום אחד (one time step)
  + לכל מפעל יש כמה פסי ייצור
  + לכל פס ייצור יש עלות תפעול (מידע פרטי)
  + אצלנו, כל פס ייצור עושה אותו דבר. אותה תהליך ייצור.
* מאחסנים מוצרים במחסנים – inventories
* הרווח נכנס ל accounts

גרף ייצור:

* הקודקודים הם מוצרים או תהליכי ייצור.
* כל תהליך ייצור מקבל קלט מוצר ומוציא פלט מוצר.

סוכנים ומו"מ:

* הסוכנים אחראים על הפרודקשן
* הסוכנים עושים מו"מ כדי להגיע להסכם לגבי מכירה/קנייה, ולאחר מכן הופכים את ההסכם לחוזה.
* ההסכם נוצר ע"י bilateral negotiations, תוך שימוש ב alternating offers protocol
* הצעה מורכבת מ:
  + קונה
  + מוכר
  + מוצר
  + כמות
  + זמן אספקה
  + מחיר ליחידה
* הסוכנים יכולים למכור רק מה שהמפעל שלהם מייצר, ולקנות רק מה שהם צריכים לפס הייצור.
* Endogenous utility functions
* על הסוכן להצמיד יוטיליטי לחוזה פוטנציאלי, בהתחשב ביכולות הייצור שלו
* פסקה לא ברורה:  
  
* הפרת חוזה = כשלקונה אין את הכסף / למוכר ת המוצר, והסימולציה לא מצליחה לממש את החוזה. מי שמפר חוזה מדווח.
* אם סוכן לא יכול לעמוד בהתחייבויות הכספיות שלו, מוכרזת עליו פשיטת רגל.
* אם סוכן לא עומד בהסכם כי אין לו את המוצרים, הוא יכול לרכוש אותם ב spot market.

Bulletin Board – מראה תוצאות בזמן אמת של כל המפעלים.

* סוכנים יכולים לקרוא את ה Bulletin Board
* אפשר לבקש עם מי לנהל מו"מ

יישויות המשחק:

* הסביבה
  + לכל מפעל f יש קווי ייצור Lf, ותהליכי ייצור M, כאשר לכל תהליך ייצור בכל פס ייצור יש עלות כלשהית.

[יוטיוב](https://www.youtube.com/watch?v=velp6T0dUno&list=PLqvs51K2Mb8IJe5Yz5jmYrRAwvIpGU2nF&index=9&t=0s) – Agent Components

**\*\*צריך להעלות את הסוכן הראשוני לאתר, לתחרות לייב**

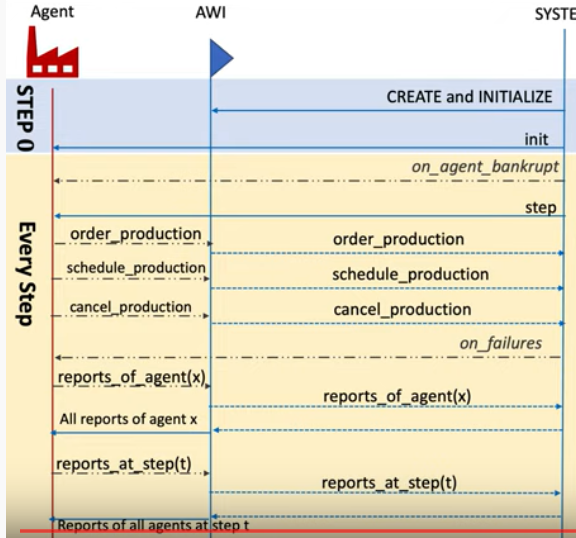
סוכן צריך ליצור negotiator או controller שישלוט בכל ה negotiators בו-זמנית.

\*\*Agent Wallet Interface – AWI, אינטרפייס דרכו לומדים על מצב העולם ופועלים בעולם.

חשוב להתייחס לדברים האלה:



Callbacks:



ארכיטקטורה של סוכנים:

* All-in-one – סוכן שעונה לכל הקולבקים.
* Component-based-Agent
  + Services – משהו שהסוכן יוצר ומשתמש. למשל: controller, negotiator, simulator (משהו שמנבא מה יקרה בעתיד)
  + Components – רכיבים שביחד בונים סוכן. אפשר לערבב את הרכיבים כדי ליצור אסטרטגיות חדשות.

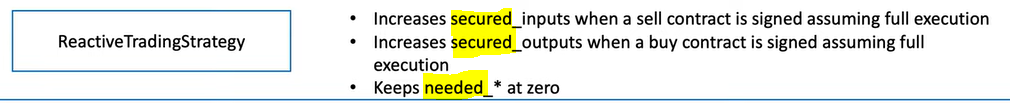
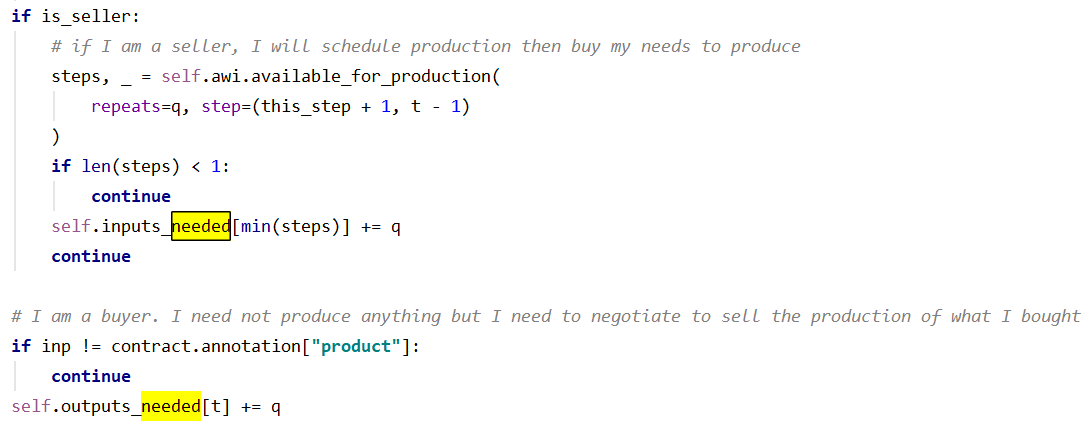
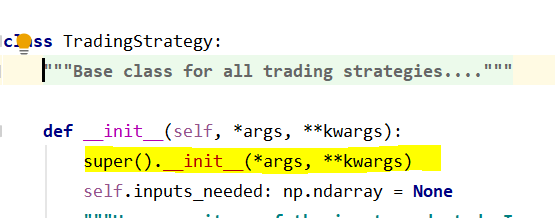
Component = מחלקה שיש לה מבנה ספציפי.

* יש פרמטרים ספציפיים ל \_\_init\_\_ ויש דיפולט
* יש לקומפוננטה דרישות מהסוכן: ממברים או פרמטרים שהקומפוננטה מצפה שממומשים אצל הסוכן
* Hooks: קולבקס שהקומפוננטה אחראית עליהם. קורא ל super כדי לאפשר קומפוננטות אחרות להשתמש בקולבק.
* Override: אם קומפוננטה דורסת מתודה, אז רק הקומפוננטה קוראת למתודה, ומונעת מקומפוננטות אחרות להשתמש בה.
* מספקות ממברים לסוכן
* Abstract: מתודות אבסטרקטיות חייבות להידרס

לא לכל הקומפוננטות יש את כל החלקים.

תמיד יש לבצע hook למתודות init, step וכל מתודה מתחילה ב on\_ (העברת מידע לסוכן) ויש לבצע דריסה לכל השאר.

הקומפוננטות המרכזיות של סוכן:

1. **Trading Strategy** – למה הסוכן מכוון; מטרת על.
   1. מספקת 4 state variable:
      1. needed\_inputs: הקלטים שצריך בכל נקודת זמן – כמות המוצרים שצריך לקנות.
      2. needed\_outputs: הפלטים שצריכים להיות ברי השגה – כמות המוצרים שצריך למכור.
      3. secured\_inputs/outputs: קלטים ופלטים חסויים – כמה כבר מכרנו וקנינו מתוך מה שצריך.
   2. סאב-קומפוננטות:
      1. Trade prediction strategy – חוזה מה יקרה בעתיד
         1. לכמה קלטים ולפלטים אני מצפה בכל time step עתידיים
         2. מה המחיר שאני מצפה שקלטים יעלו
         3. המחיר שבו אמכור את הפלטים (צפי)
      2. Execution rate prediction strategy – מספק מתודה לחיזוי כמויות. לכל חוזה, כמה כמות בחוזה הזה כנראה תצא לפועל (*predicted quality*).
      3. Signing Strategy – מקבל את ה *agreements* (ואת ה *predicted quality*) מה negotiation manager, ומפעיל אסטרטגיית חתימה כדי ליצור את ה contract.   
         ב contracts משתמשים כל הקומפוננטות האחרות.
   3. שולט ב overall strategy של הסוכן: מה הוא רוצה לקנות, למכור, ועוקב אחר כמה מזה הושג.
   4. מימושים:
      1. NoTradingStrategy – הכל 0 (כמה צריך וכמה השגתי). הסוכן לא רוצה לעשות כלום.
      2. ReactiveTradingStrategy – מניח full execution: יעלה את ה-secured\_inputs כאשר חוזה מכירה נחתם, יעלה את ה- secured\_outputs כאשר חוזה קנייה נחתם.   
          ה- needed תמיד = 0. כלומר אני לא צריך כלום ותמיד יש אקסטרה.  
           
         **\*\*\*זה כנראה הפוך**, כי בקוד:  
         
      3. PredictionBasedTradingStrategy – מעדכן את ה secured quantities (קניות ומכירות) כאשר הוא יודע שחוזה ייחתם – בהתבסס על ERPS.  
         רכיבים:
         1. TradePredictionStrategy = עושה אתחול ל needed  
            **\*\*\*איפה זה קורה??**   
            **\*\*\*לאן ה super הזה הולך?**  
              
            1. חוזה מו"מים עתידיים: לכמה קלטים לצפות באיזה מחיר בכל time step  
               **\*\*\*לממש ML אוןליין??**
            2. מחלקה שמממשת: FixedTradePredictionStrategy – חוזה מספרים קבועים עבור ה expected number (פלטים וקלטים) ועבור ה cost/price (פלטים וקלטים)
         2. ExecutionRatePredictionStrategy = חוזה את ה fraction של כל חוזה להיחתם.
            1. חוזה כמה יחידות בחוזה הולכות לביצוע. אפשר להשתמש ב financial reports, וב (???). **אפשר גם להשתמש ב ML פה**.
            2. מימוש: FixedERPStrategy – מנבא שבר קבוע לכל החוזים
            3. מימוש: MeanERPStrategy – חוזה את ה fraction הממוצע מכל החוזים הקודמים, אוןליין. ה prior שלו הוא 0.5 וזה מתעדכן עם כל חוזה.  
               **\*\*אפשר לנסות לבדוק כמה שברים, למצוא את הכי טוב ולהשתמש בו בתחרות**
2. **Negotiation Manager** – להשיג את המטרה של (1) דרך מו"מ.
   1. צריך להגיד לו מה ה Target quantity וה- Acceptable unit price.
      1. Target quantity = needed (כמה מוצרים צריך לקנות / למכור בכללי) פחות secured (כמה כבר קנינו/ מכרנו). ערך זה צריך להיות מעודכן לכל time step
   2. יוצר קונטרולרים ומנהלי משא ומתן כך שישיגו את המטרה שהגדרנו לו ב (a), דרך agreements.
   3. הקומפוננטות הקיימות במערכת:
      1. IndependentNegotiationsManager – בהינתן פונקציית יוטיליטי, הקומפוננטה משתמשת בה במו"מים עצמאיים. כל מו"מ הוא עצמאי; לא יודע על מו"מים אחרים. יש מצב כאן ל over-contracts
      2. MovingRangeNegotiationManager – יוצר קונטרולר לכל המכירות וקונטרולר לכל הקניות.   
          הקונטרולרים שומרים את ה negotiators בסנכרון כדי שלא יהיה over-contracts. לוקח בחשבון רק חוזים פוטנציאלים בצעדים הקרובים.   
          משתמש ב SyncController שמשתמש בפונקציית יוטיליטי לינארית פשוטה.
      3. StepNegotiationManager – בכל time step יוצר קונטרולר לקניות וקונטרולר למכירות.   
          משתמש ב StepController שמשתמש באותו אלגוריתם מו"מ (?)
   4. המטרה: להתחיל מו"מ פרואקטיבי במהלך step, ולענות על חתימת חוזים, ומשתמש ב Acceptable unit price כדי לקבוע אג'נדה למו"מים.
3. **Signing Strategy** – המנהל (2) משתמש בזה כדי להחליט על מה לחתום.
   1. מחליט על אילו חוזים לחתום.
   2. מימושים:
      1. SignAll – חותם על הכל.
      2. SignAllPossible – חותם על מה שריאלי.
4. **Production Strategy** – אומר למפעל מה לייצר. מחליט מה צריך לייצר ע"י hooking לפונקציית step.
   1. להסתכל ב scml2020.components.production
   2. 3 אסטרטגיות מרכזיות במערכת:
      1. SupplyDrivenProductionStrategy – להפיק כמה שיותר, כמה שאפשר. כשיש את המוצרים, אז תפעיל את פס הייצור. הקלטים ישר מועברים לפלטים.
         1. חסרון: סיכון ליותר מדי ייצור, ללא היכולת למכור את זה בעתיד. מאבדים את ה production cost.
      2. DemandDrivenProductionStrategy – מייצר כמות שתספיק לכל החוזים החתומים, בהנחה שהם ייצאו לפועל. *\*\*אפשר לשפר את זה ע"י פרדיקציה של כמות הוצאה לפועל, ולא להניח שהחוזים ייצאו לפועל.*
      3. TradeDrivenProductionStrategy – כמו ii אבל מתעלם מחוזים שהסוכן יזם.   
         

הטוטוריאל השני

סיכום:

לכל סוכן צריך שיהיה:

1. Trading Strategy קומפוננטה. ניתן להיעזר ב prediction / execution rate prediction / signing אסטרטגיות
2. **Negotiation manager** קומפוננטה. יוצר negotiators ו controllers.
3. Production Strategy

יוטיוב – Basic SAO Controllers and Negotiators

NegMAS Documentation

**\*לקרוא את ה overview ולעבור על ה tutorials**

<http://yasserm.com/negmas/>

Base modules – outcomes & utilities

* יש פה פונקציות יוטיליטי

...

\*\*אפשר להשתמש ב genius negotiator

בייס קלאס של הכל: NamedObject

יש 3 אובייקטים רציונלים שיש להם פונקציית יוטיליטי: agent, controller, negotiator

* אנחנו מממשים סוכן.
* מו"מ: יש negotiators שהם כבר מוגדרים ואנחנו יכולים להשתמש בהם, או להגדיר אסטרטגייה חדשה משלנו.
* קונטרולרים: יש קונטרולרים שאפשר להשתמש בהם ויש כאלה שכבר קיימים.

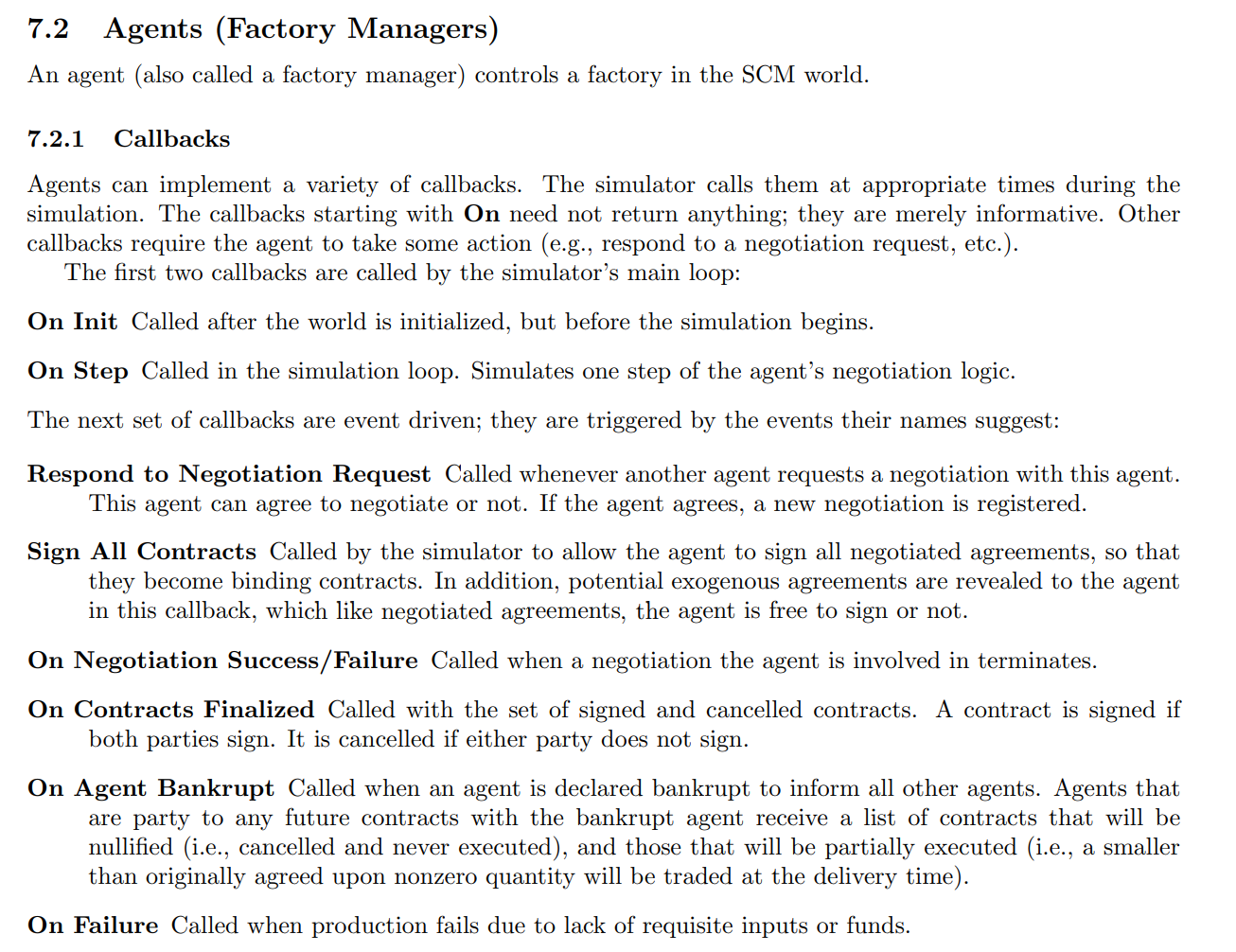
לכל סוכן יש awi שמחבר אותו לעולם world.

ה negotiators מדברים זה עם זה דרך mechanism, וצריך להיות להם ufunc.

<http://www.yasserm.com/scml/scml2020.pdf>

אינטרפייס:

קולבקים – הסימולציה קוראת להם



פעולות – הסוכן יוזם אותן

